

Algorithmica Research AB

Running a Monte-Carlo simulation on a Quantlab cluster using Quantlab 3.0

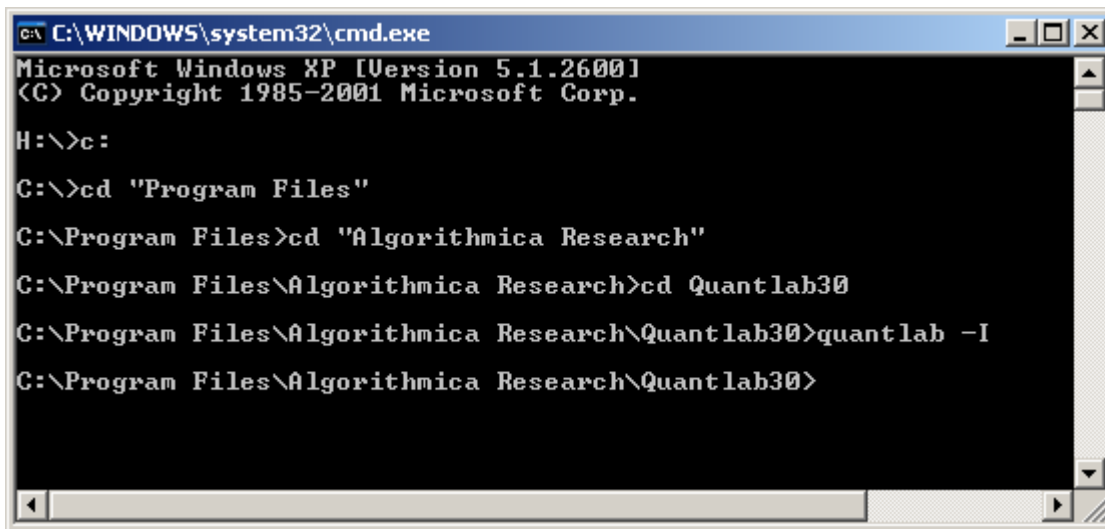
Running a Monte-Carlo simulation on a Quantlab cluster

This is an example of how to create a mini-cluster on separate servers/desktop pc:s and then executing a distributed call to them to price an Asian option using Monte-Carlo simulation.

Simple installation

We will assume that all servers that should run a Quantlab instance will have a valid licence key and a proper stand-alone Quantlab installed somewhere.

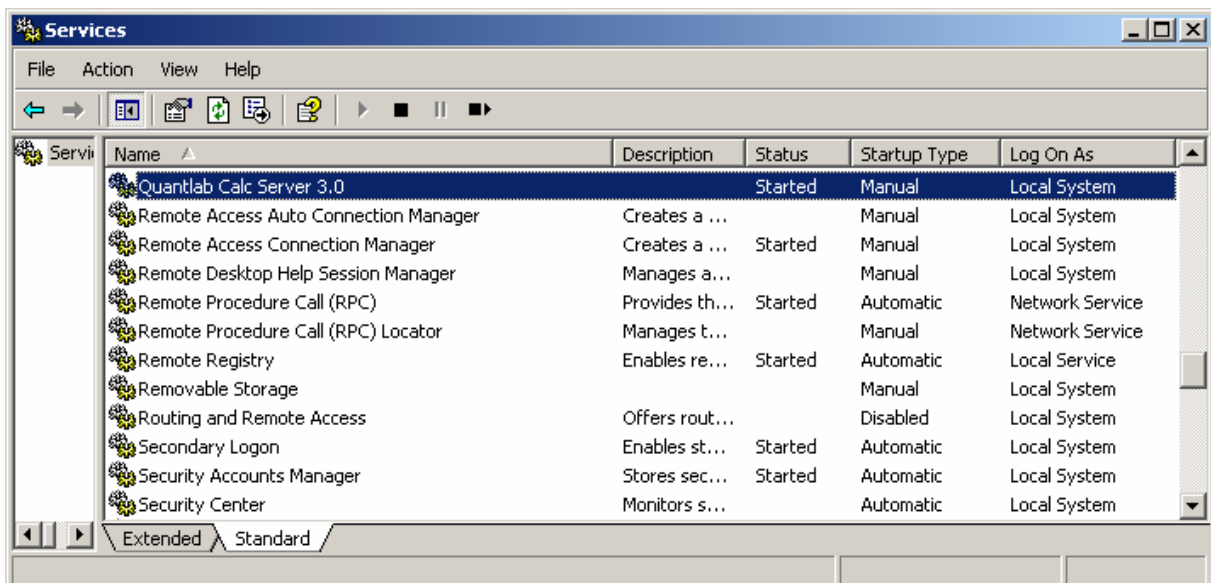
To install the “QLC”, the Quantlab Calc Server, simply write `quantlab -I` from the command prompt on the server that should have the QLR running as a service. First you need to find the place where the Quantlab.exe is installed.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>c:
C:\>cd "Program Files"
C:\Program Files>cd "Algorithmica Research"
C:\Program Files\Algorithmica Research>cd Quantlab30
C:\Program Files\Algorithmica Research\Quantlab30>quantlab -I
C:\Program Files\Algorithmica Research\Quantlab30>
```

Now go to the services manager and start the service. Ensure that a user with the correct credentials is logged-on as the service user.



Now we are ready to access all Quantlab Calc Servers that have gone through the same installation procedure.

A simplistic Monte-Carlo simulation

This example is a simplistic Monte-Carlo simulation to get the price of a path-dependent (arithmetic average) European option.

The idea is to treat the total number of simulations as a number of independent simulations carried out on n-servers, each running total / n simulations. When all servers are finished we simply average over the n-result sets. In order for this simple trick to work we need to ensure that the statistical properties will not be changed.

To ensure that we have a unique run of random numbers we can seed the random number generator on each server with a different starting seed. Here we will use the internal clock's milliseconds to seed the random number function. An alternative would be to fixate the starting seeds in order to have a reproducible result.

There are two ways of getting the function to execute onto the remote servers. You can install the function locally on each server as a library file (which would be the normal case). But you may also send a string containing the complete code at run-time. It will then be compiled when the connection to the remote server is done. In this example we use this option for simplicity.

Let's look at the code:

```

//First we list all our servers by name - here we have 4 qlc in place
vector(string) v_servers = ['lincoln', 'jefferson', 'roosevelt', 'washington'] ;

//Second we concatenate a string with the simulation code function
//Luckily I know that it works already ...
string myQlang = strcat([
    'out number myDiscOptionPayoff(logical iscall, number strike, ',
    'number S0, number time, ',
    'number rate, number carry, number vol, ',
    'number steps, number sim){ ',
    'rng r = rng(millisecond(now())); ',
    'number dt = time / steps; ',
    'number drift = (carry - vol*vol / 2) * dt; ',
    'number dt_vol_sqr = vol * sqrt(dt); number z; ',
    'if(iscall){ z=1 ;} else {z=-1;} ',
    'number Payoffsum = 0;',
    'for(number s = 1; s <=sim; s++){ ',
    'number St = S0; ',
    'number Ssum = 0; ',
    'for(number j = 1; j <=steps; j++) { ',
    'St = St * exp(drift + dt_vol_sqr * r.gauss());',
    'Ssum = Ssum + St; } ',
    'number Savg = Ssum / steps;',
    'Payoffsum = Payoffsum + max(z * (Savg - strike), 0); }',
    'return exp(-rate * time)*Payoffsum/sim;}]');

//Here is the function that will assign the call to the cluster
//It has a number of input arguments so that the user can change the settings

out vector(number) clusterOptionPrice(logical iscall, number strike, number S0, number time, number
rate, number carry, number vol, number steps, number sim)
{
    timestamp start = now() ; //To time our tests

    //we connect to the servers and pass the function as a string argument
    vector(qlc.connection) v_conn = qlc.connect(v_servers, myQlang);
    number localsim = sim/v_size(v_servers);

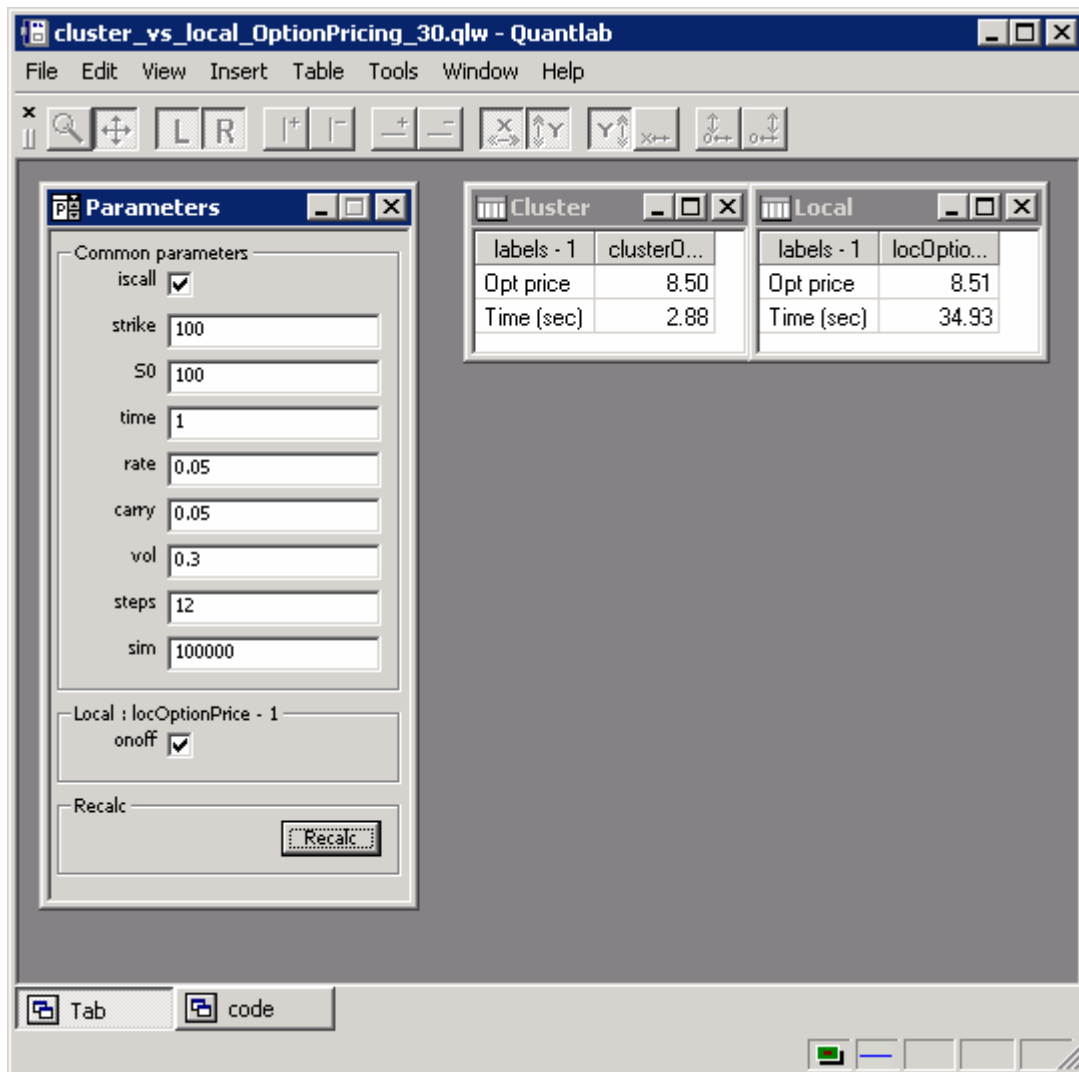
    //using our vector of connections, we start the evaluation of the function
    //we pass the input arguments as variants to the remote servers
    v_conn.start_eval('myDiscOptionPayoff',[variant(iscall), variant(strike), variant(S0),variant(time),
        variant(rate),variant(carry), variant(vol), variant(steps), variant(localsim)]);

    //now it's just to wait for the servers to finish and sum the results
    //the return is also a variant so we have to use the get_number()
    number ret = v_sum(qlc.multi_wait(v_conn).get_number())/v_size(v_servers);

    timestamp stop = now() ;
    number duration = (stop - start)/1000 ;
    return [ret,duration];
}

```

In order for us to get a user interface we attach the clusterOptionPrice function in table. The input arguments appear as controls. For comparative reasons we have also taken an identical version of the function that run locally in order to get a sense of the speed gains from the cluster.



Here we are pricing an average option with 1 year maturity and monthly averaging points. We run 100000 simulations in total having 12 steps per simulation. The performance gain is of a magnitude of 10:1 compared with running it on the stand-alone single desktop.

Why is it not exactly 4 times faster? The local desktop pc is a much slower machine to start with. Can we do even better?

Yes. Since we know that the remote servers are running a dual core processor we can just add another call to the same servers in our `v_servers` vector. This will give us an additional calculation thread per server. How large the speed up will be obviously depends on if we get both core's full attention or not.

```
//First we list all our servers by name - now we get two threads on each server
vector(string) v_servers = ["lincoln", "jefferson", "roosevelt", "washington",
                           "lincoln", "jefferson", "roosevelt", "washington"] ;
```

cluster_vs_local_OptionPricing_30.qlw - Quantlab

File Edit View Insert Table Tools Window Help

Parameters

Common parameters

iscall

strike 100

S0 100

time 1

rate 0.05

carry 0.05

vol 0.3

steps 12

sim 100000

Local : locOptionPrice - 1

onoff

Recalc

Recalc

Cluster

| labels - 1 | clusterO... |
|------------|-------------|
| Opt price | 8.51 |
| Time (sec) | 2.49 |

Local

| labels - 1 | locOptio... |
|------------|-------------|
| Opt price | 8.53 |
| Time (sec) | 34.90 |

Tab code